Spring Cloud로 개발하는 마이크로서비스 애플리케이션

목차

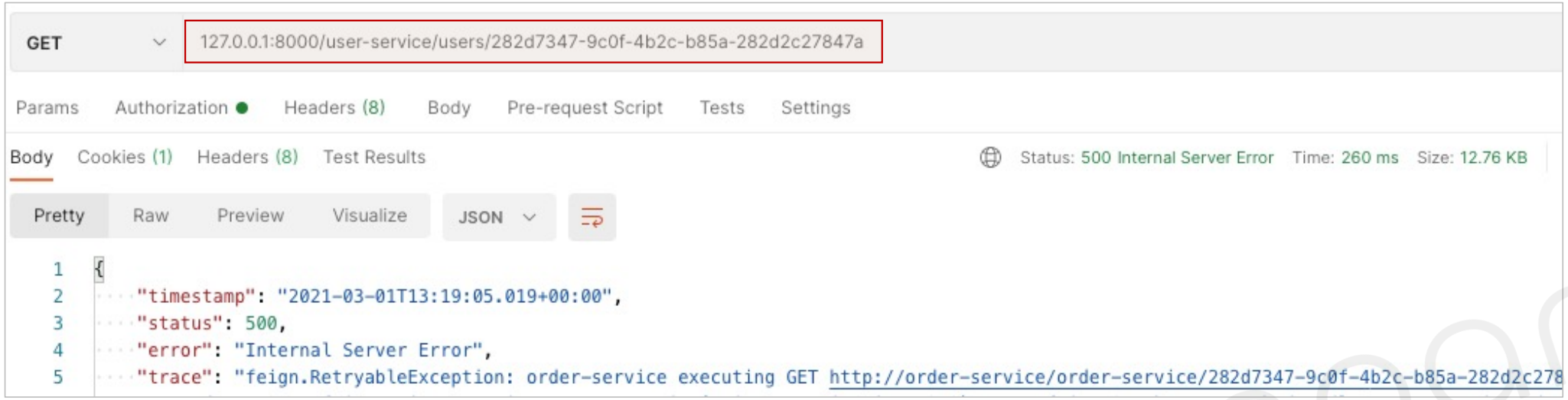
Part II

Section 13.

장애 처리와 Microservice 분산 추적

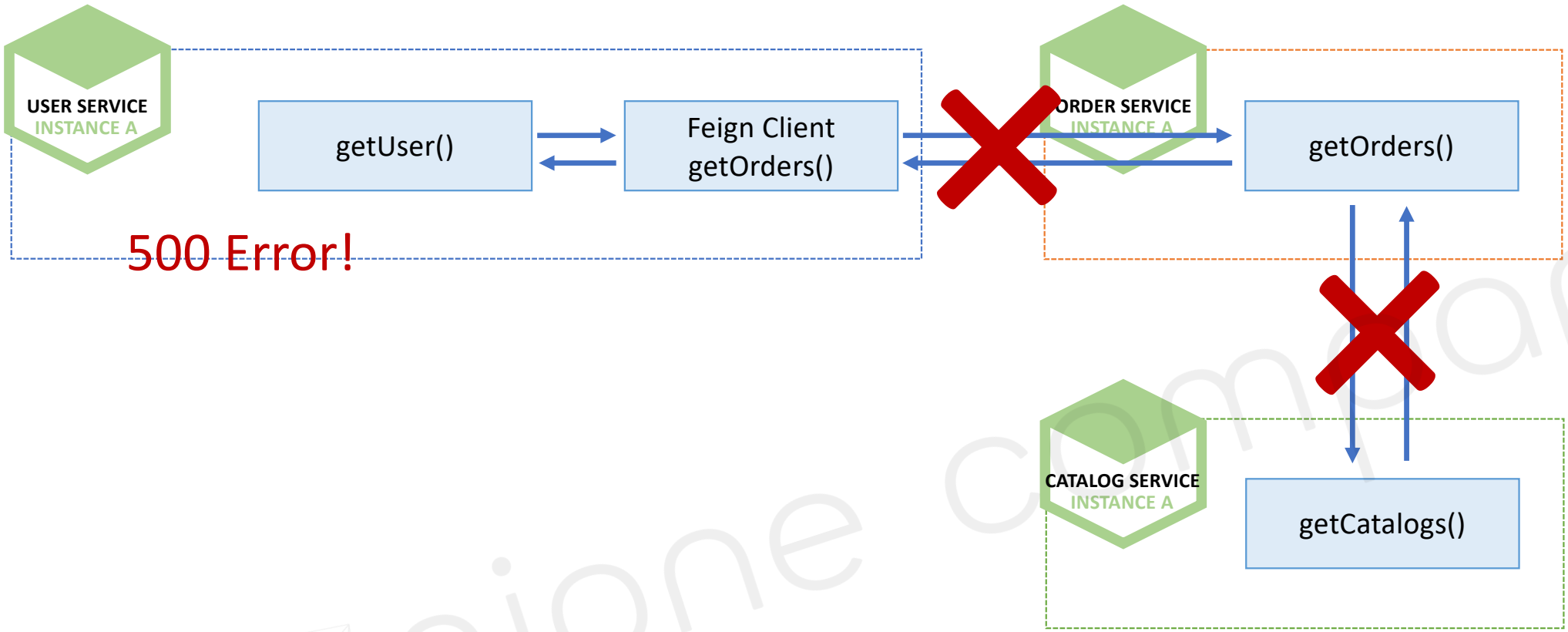
- CircuitBreaker
- Resilience4j
- Distributed Tracing
- Trace ID and Span ID
- Zipkin server 활용

Microservice 통신 시 연쇄 오류

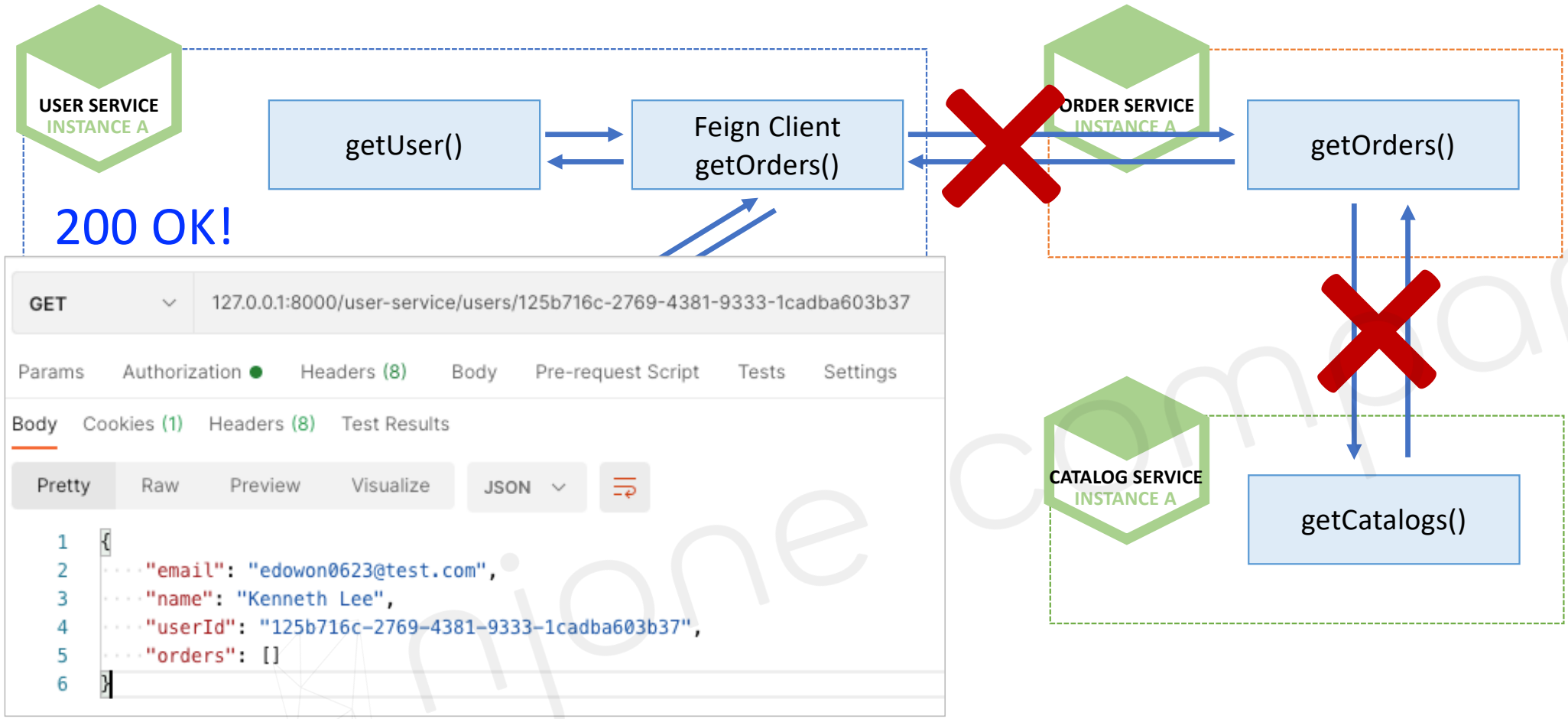



```
java.net.UnknownHostException Create breakpoint : order-service
    at java.base/sun.nio.ch.NioSocketImpl.connect(NioSocketImpl.java:567) ~[na:na]
    at java.base/java.net.Socket.connect(Socket.java:648) ~[na:na]
    at java.base/sun.net.NetworkClient.doConnect(NetworkClient.java:177) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:474) ~[na:na]
```

Microservice 통신 시 연쇄 오류

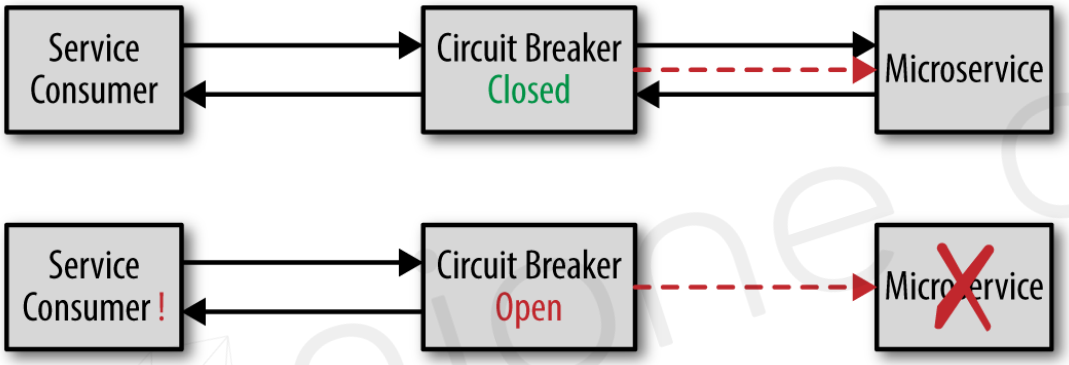


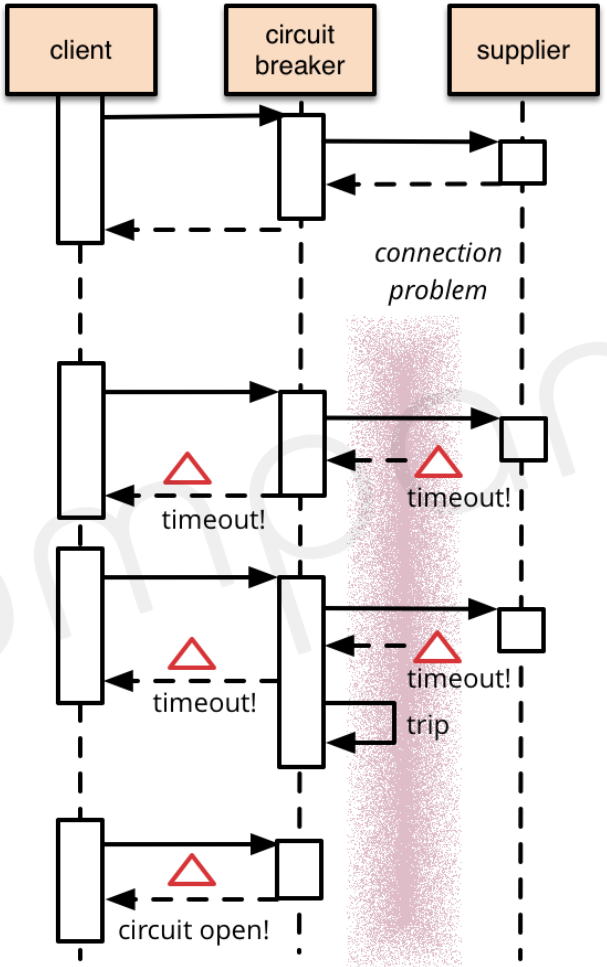
Microservice 통신 시 연쇄 오류



CircuitBreaker

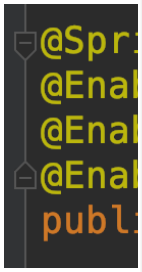
- CircuitBreaker
  - https://martinfowler.com/bliki/CircuitBreaker.html
  - 장애가 발생하는 서비스에 반복적인 호출이 되지 못하게 차단
  - 특정 서비스가 정상적으로 동작하지 않을 경우 다른 기능으로 대체 수행 → 장애 회피





Spring Cloud Netflix Hystrix

- Spring Cloud Netflix Hystrix



CURRENT	REPLACEMENT
Hystrix	Resilience4j
Hystrix Dashboard / Turbine	Micrometer + Monitoring System
Ribbon	Spring Cloud Loadbalancer
Zuul 1	Spring Cloud Gateway
Archaius 1	Spring Boot external config + Spring Cloud Config

Replacements technologies. Taken from [1]

er-ws)

r-ws)

Resilience4j

- Resilience4j
  - resilience4j-circuitbreaker: Circuit breaking
  - resilience4j-ratelimiter: Rate limiting
  - resilience4j-bulkhead: Bulkheading
  - resilience4j-retry: Automatic retrying (sync and async)
  - resilience4j-timelimiter: Timeout handling
  - resilience4j-cache: Result caching
  - https://resilience4j.readme.io/docs/getting-started
  - https://github.com/resilience4j/resilience4j

Introduction

Resilience4j is a lightweight, easy-to-use fault tolerance library inspired by Netflix Hystrix, but designed for Java 8 and functional programming. Lightweight, because the library only uses Vavr, which does not have any other external library dependencies. Netflix Hystrix, in contrast, has a compile dependency to Archaius which has many more external library dependencies such as Guava and Apache Commons Configuration.

Resilience4j – CircuitBreaker

- DefaultConfiguration

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
</dependency>
```

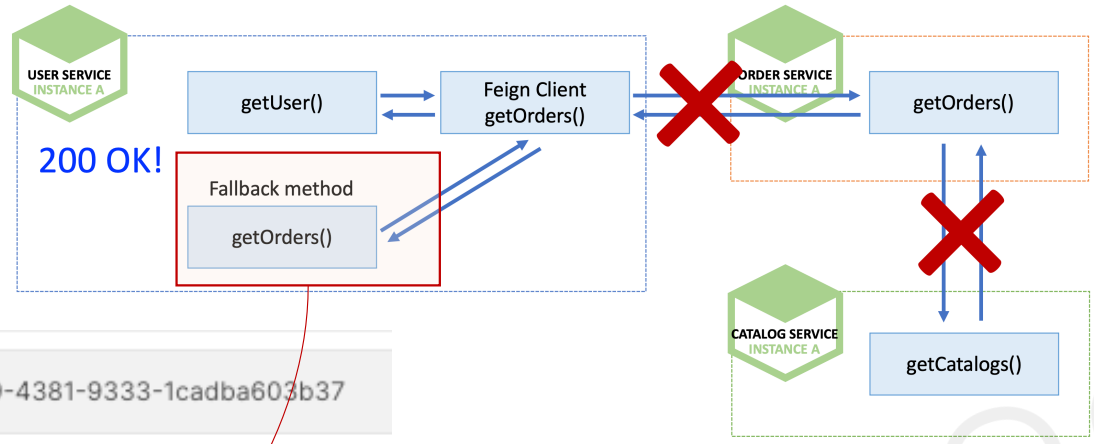
pom.xml

```
@Autowired
CircuitBreakerFactory circuitBreakerFactory;

        /* ErrorDecoder */
//      List<ResponseOrder> ordersList = orderServiceClient.getOrders(userId);

        /* CircuitBreaker */
        CircuitBreaker circuitBreaker = circuitBreakerFactory.create("circuitbreaker");
        List<ResponseOrder> ordersList = circuitBreaker.run(() -> orderServiceClient.getOrders(userId),
                throwable -> new ArrayList<>());

        userDto.setOrders(ordersList);
```

UserServiceImpl.java

Resilience4j – CircuitBreaker

- DefaultConfiguration



GET 127.0.0.1:8000/user-service/users/125b716c-2769-4381-9333-1cadba603b37

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Body Cookies (1) Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "email": "edowon0623@test.com",
    "name": "Kenneth Lee",
    "userId": "125b716c-2769-4381-9333-1cadba603b37",
    "orders": []
}
```

Resilience4j – CircuitBreaker

- Customize CircuitBreakerFactory → ***Resilience4JCircuitBreakerFactory***

```
@Configuration
public class Resilience4JConfiguration {
    @Bean
    public Customizer<Resilience4JCircuitBreakerFactory> globalCustomConfiguration() {

        // the circuitBreakerConfig and timeLimiterConfig objects
        CircuitBreakerConfig circuitBreakerConfig = CircuitBreakerConfig.custom()
                .failureRateThreshold(4)
                .waitDurationInOpenState(Duration.ofMillis(1000))
                .slidingWindowType(CircuitBreakerConfig.SlidingWindowType.COUNT_BASED)
                .slidingWindowSize(2)
                .build();
```

- CirucitBreaker를 실행 할지(Open) 결정하는 failure rate threshold percentage
- default: 50

- CircuitBreaker를 open한 상태를 유지하는 지속 기간을 의미
- 이 기간 이후에 Half-open 상태
- default: 60seconds

- CircuitBreaker가 종료(Close) 되기 위한 조건을 결정하는 타입
- COUNT_BASED or TIME_BASED

- CircuitBreaker가 종료(Close) 되기 위한 조건 값의 크기
- default: 100

Resilience4j – CircuitBreaker

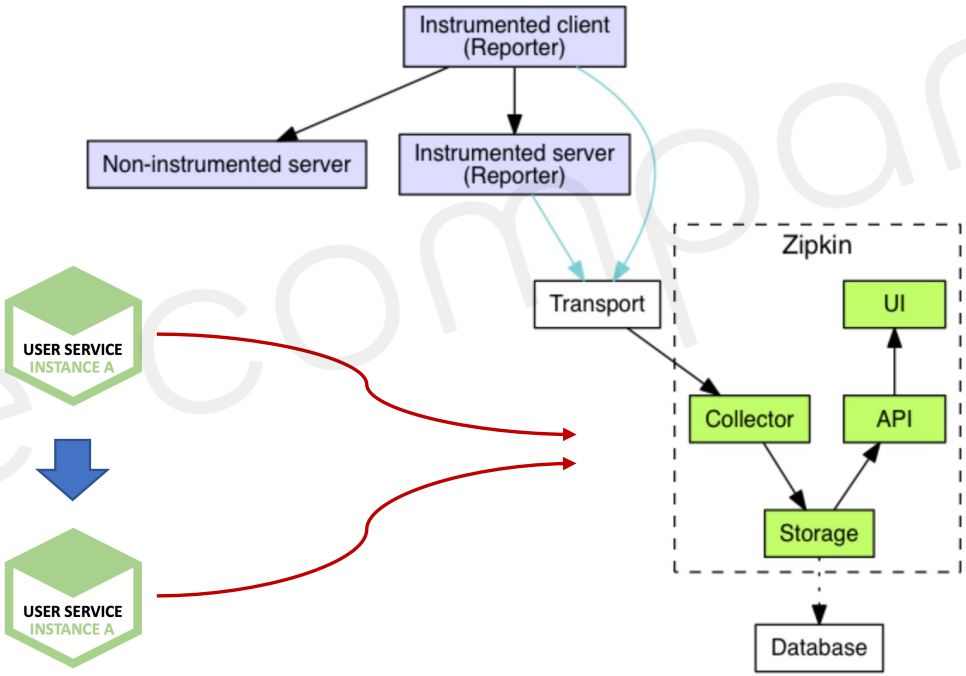
- Customize CircuitBreakerFactory → ***Resilience4JCircuitBreakerFactory***

```
        TimeLimiterConfig timeLimiterConfig = TimeLimiterConfig.custom()
                .timeoutDuration(Duration.ofSeconds(4))
                .build();

        return factory -> factory.configureDefault(id -> new Resilience4JConfigBuilder(id)
                .timeLimiterConfig(timeLimiterConfig)
                .circuitBreakerConfig(circuitBreakerConfig)
                .build());
    }
}
```

- TimeLimiter는 future supplier의 time limit을 정하는 API
- default: 1초

Microservice 분산 추적

- Zipkin
  - https://zipkin.io/
  - Twitter에서 사용하는 분산 환경의 Timing 데이터 수집, 추적 시스템 (오픈소스)
  - Google Drapper에서 발전하였으며, 분산환경에서의 시스템 병목 현상 파악
  - Collector, Query Service, Databasem WebUI로 구성
  - ***Span***
    - 하나의 요청에 사용되는 작업의 단위
    - 64 bit unique ID
  - ***Trace***
    - 트리 구조로 이뤄진 Span 셋
    - 하나의 요청에 대한 같은 Trace ID 발급



Microservice 분산 추적

- Spring Cloud Sleuth
  - 스프링 부트 애플리케이션을 Zipkin과 연동
  - 요청 값에 따른 Trace ID, Span ID 부여
  - Trace와 Span Ids를 로그에 추가 가능
    - servlet filter
    - rest template
    - scheduled actions
    - message channels
    - feign client

Spring Cloud Sleuth 3.0.1

OVERVIEW LEARN SAMPLES

Spring Cloud Sleuth provides Spring Boot auto-configuration for distributed tracing.

Features

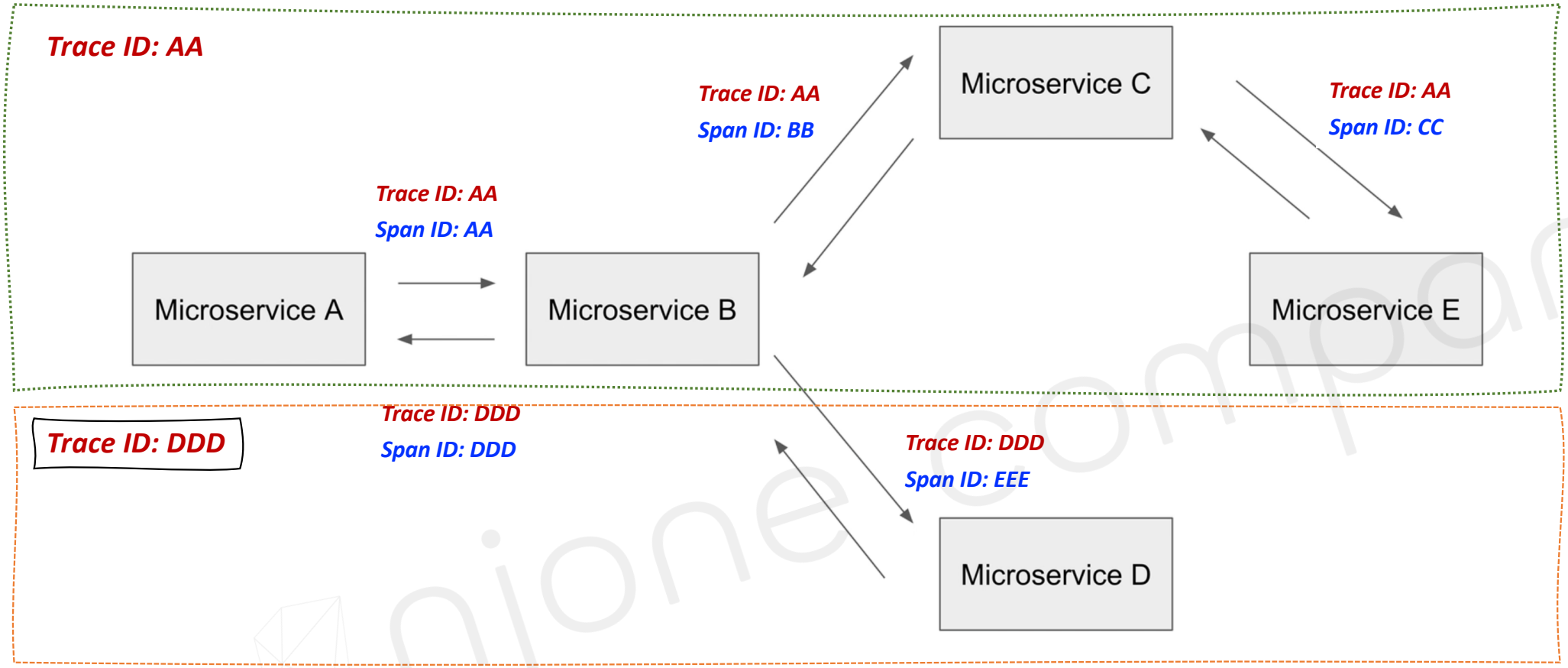
Sleuth configures everything you need to get started. This includes where trace data (spans) are reported to, how many traces to keep (sampling), if remote fields (baggage) are sent, and which libraries are traced.

Specifically, Spring Cloud Sleuth...

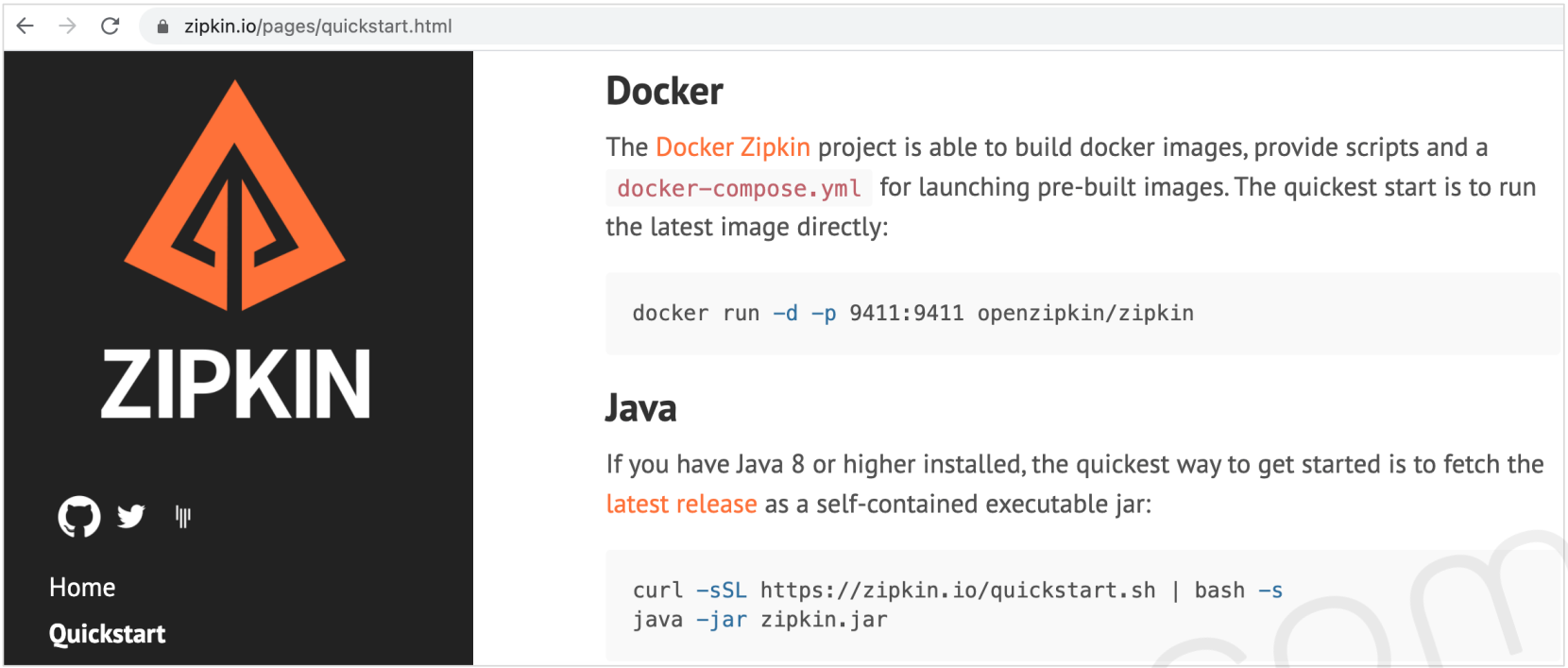
- Adds trace and span ids to the Slf4J MDC, so you can extract all the logs from a given trace or span in a log aggregator.
- Instruments common ingress and egress points from Spring applications (servlet filter, rest template, scheduled actions, message channels, feign client).
- If `spring-cloud-sleuth-zipkin` is available then the app will generate and report Zipkin-compatible traces via HTTP. By default it sends them to a Zipkin collector service on localhost (port 9411). Configure the location of the service using `spring.zipkin.baseUrl`.

Microservice 분산 추적

- Spring Cloud Sleuth + Zipkin



Zipkin server 설치

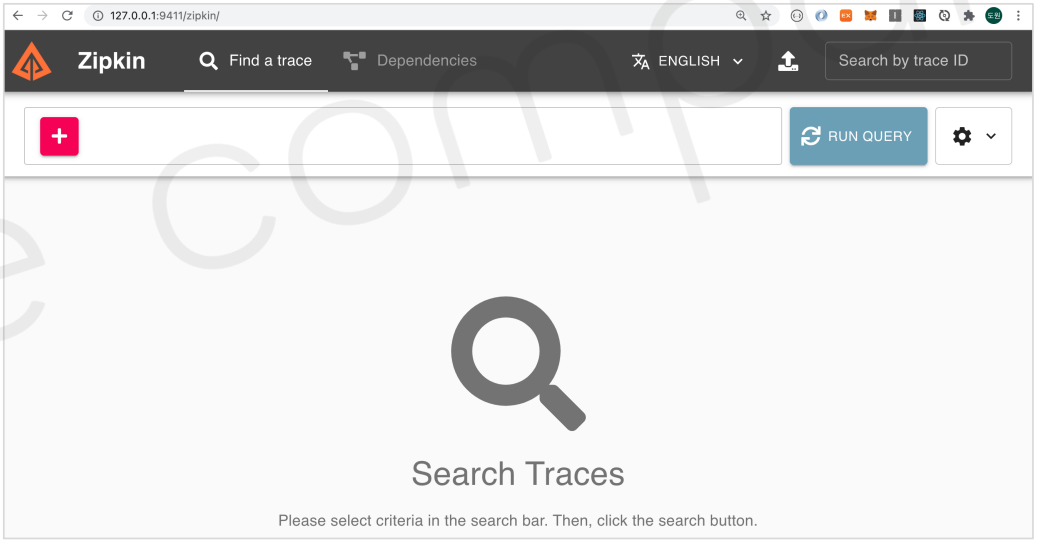



```
curl -sSL https://zipkin.io/quickstart.sh | bash -s
Thank you for trying Zipkin!
This installer is provided as a quick-start helper, so you can try Zipkin out
without a lengthy installation process.

Fetching version number of latest io.zipkin:zipkin-server release...
Latest release of io.zipkin:zipkin-server seems to be 2.23.2

Downloading io.zipkin:zipkin-server:2.23.2:exec to zipkin.jar...
```

Zipkin server 기동

```
java -jar zipkin.jar
```

```
                  oo
                 oooo
                oooooo
               oooooooo
              oooooooooo
             oooooooooooo
           oooooooo  oooooooo
          oooooo      oooooooo
         oooooo        oooooooo
        oooooo    o  o    oooooo
       oooooo    oo  oo    oooooo
     ooooooo  oooo  oooo  ooooooo
    oooooo    ooooo  ooooo  ooooooo
   oooooo    oooooo  oooooo  ooooooo
  oooooooo      oo  oo      oooooooo
  ooooooooooooo oo  oo ooooooooooooo
     oooooooooooo    oooooooooooo
         oooooooo  oooooooo
             oooo  oooo


   _______ ____   _  _____ _   _
  |__   /_ _|  _ \| |/ /_ _| \ | |
    / / | || |_) | ' / | ||  \| |
   / /_ | ||  __/| . \ | || |\  |
  |____|___|_|   |_|\_\___|_| \_|

:: version 2.23.2 :: commit 7bf3aab ::

2021-03-02 01:14:25.648  INFO [/] 87530 --- [oss-http-*:9411] c.l.a.s.Server
                : Serving HTTP at /[0:0:0:0:0:0:0:0]:9411 - http://127.0.0.1:9411/
```



Users Microservice 수정

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zipkin</artifactId>
    <version>2.2.3.RELEASE</version>
</dependency>
```

pom.xml

```yaml
spring:
  application:
    name: user-service
zipkin:
  base-url: http://localhost:9411
  enabled: true
sleuth:
  sampler:
    probability: 1.0
```

application.yml

```java
/* CircuitBreaker */
log.info("Before call orders microservice");
CircuitBreaker circuitBreaker = circuitBreakerFactory.create("my_circuit_breaker");
List<ResponseOrder> ordersList = circuitBreaker.run(() -> orderServiceClient.getOrders(userId),
        throwable -> new ArrayList<>());
log.info("after called orders microservice");

userDto.setOrders(ordersList);
```

UsersServiceImpl.java

Trace ID and Span ID

POST http://127.0.0.1:8000/order-service/a09741d4-b5c4-44fb-a9a6-8a6312e9b58c/orders

Params Authorization Headers (11) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
    "productId": "CATALOG-003",
    "qty": 5,
    "unitPrice": 1500
}
```

Body Cookies (1) Headers (3) Test Results Status: 201 Created

Pretty Raw Preview Visualize JSON

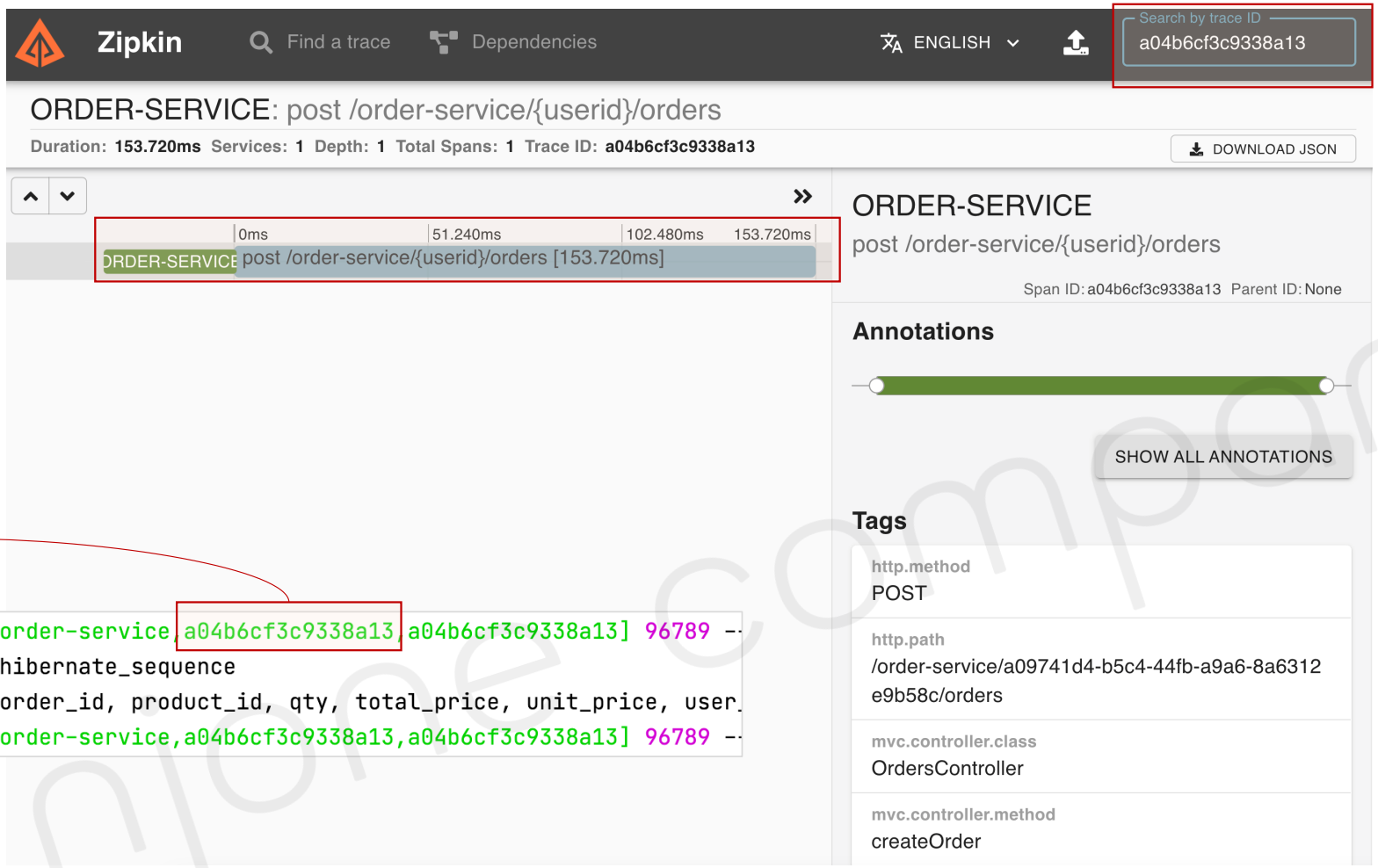
```
{
    "productId": "CATALOG-003",
    "qty": 5,
    "unitPrice": 1500,
    "totalPrice": 7500,
    "orderId": "6f3bed3b-0ba4-483b-83b1-2358b3bba500"
}
```

Trace ID Span ID

```
2021-03-02 23:47:35.314  INFO [order-service,a04b6cf3c9338a13,a04b6cf3c9338a13] 96789 --
Hibernate: call next value for hibernate_sequence
Hibernate: insert into orders (order_id, product_id, qty, total_price, unit_price, user
2021-03-02 23:47:35.413  INFO [order-service,a04b6cf3c9338a13,a04b6cf3c9338a13] 96789 --
```

```
[o-auto-1-exec-1] c.e.o.controller.OrdersController        : Before add orders data

d, id) values (?, ?, ?, ?, ?, ?, ?)
[o-auto-1-exec-1] c.e.o.controller.OrdersController        : After added orders data
```

Trace ID and Span ID



Trace ID

```
2021-03-02 23:47:35.314  INFO [order-service,a04b6cf3c9338a13,a04b6cf3c9338a13] 96789 --
Hibernate: call next value for hibernate_sequence
Hibernate: insert into orders (order_id, product_id, qty, total_price, unit_price, user
2021-03-02 23:47:35.413  INFO [order-service,a04b6cf3c9338a13,a04b6cf3c9338a13] 96789 --
```

Trace ID and Span ID

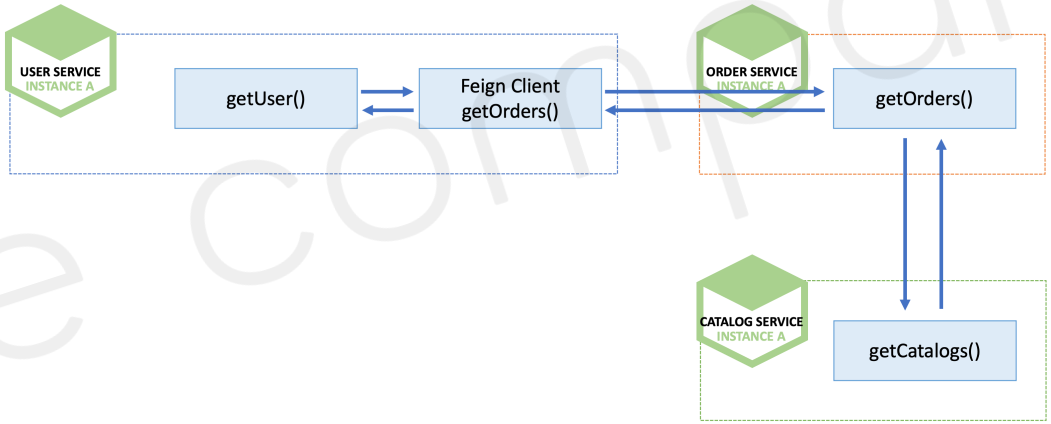
GET 127.0.0.1:8000/user-service/users/a09741d4-b5c4-44fb-a9a6-8a6312e9b58c

Params Authorization Headers (8) Body Pre-request Script Tests Settings

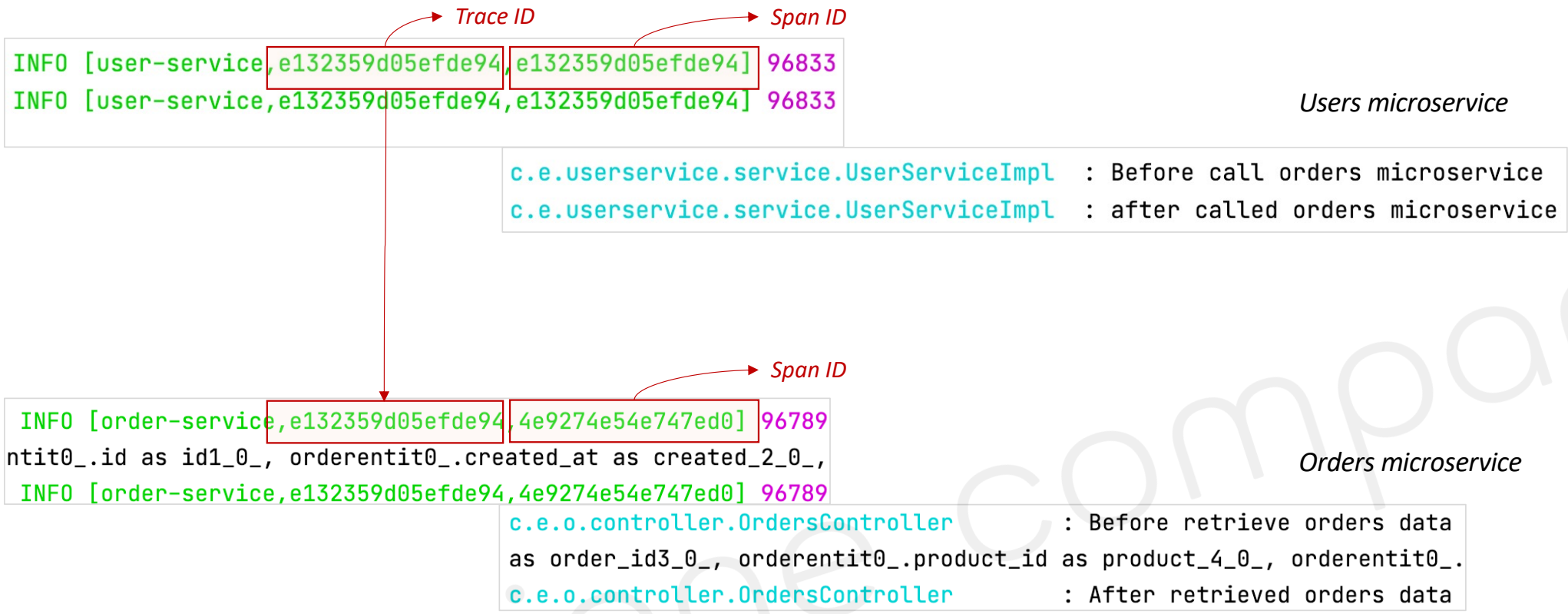
Body Cookies (1) Headers (8) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

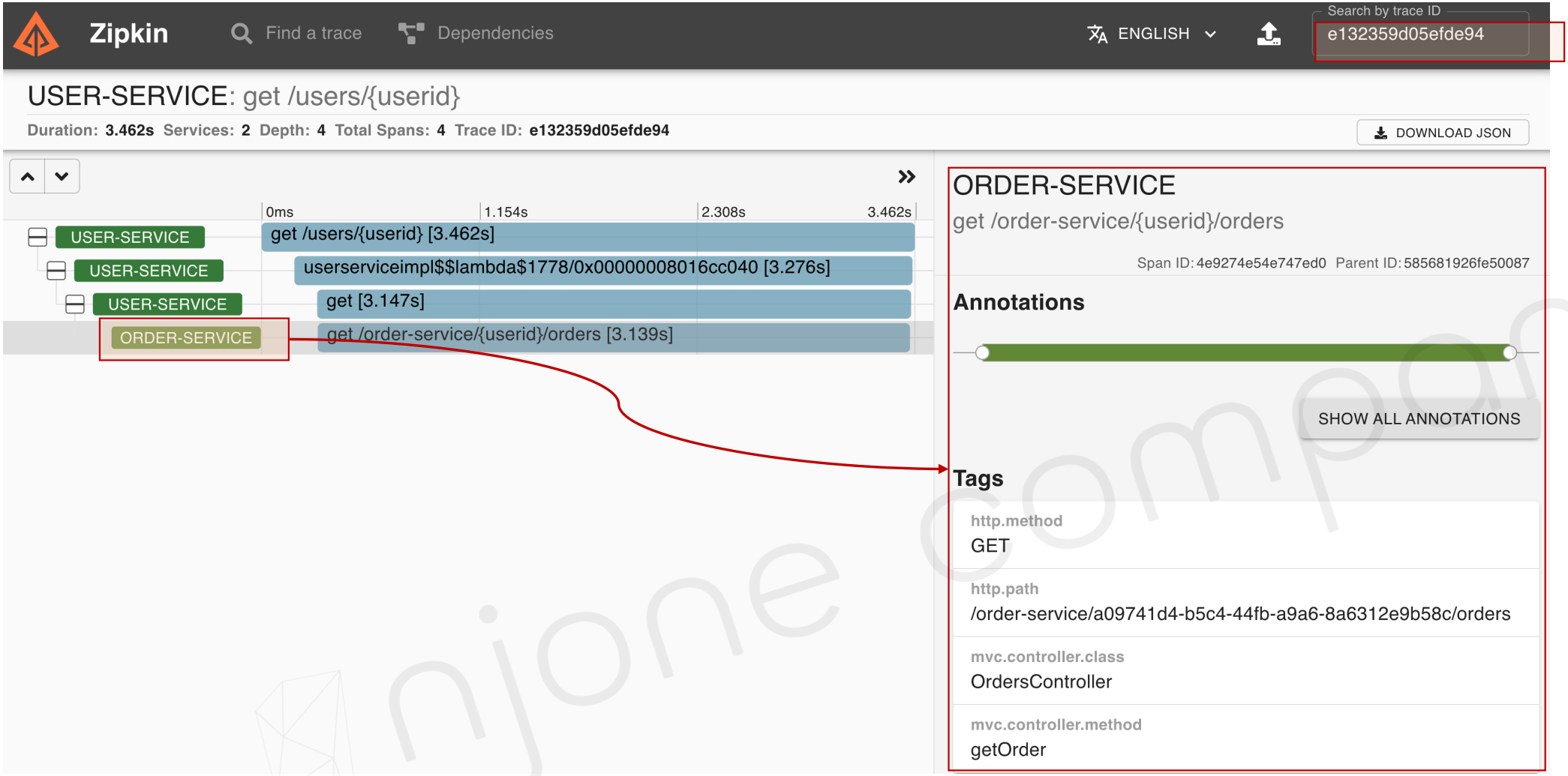

```
{
    "email": "edowon0623@test.com",
    "name": "Kenneth Lee",
    "userId": "a09741d4-b5c4-44fb-a9a6-8a6312e9b58c",
    "orders": [
        {
            "productId": "CATALOG-004",
            "qty": 10,
            "unitPrice": 1500,
            "totalPrice": 15000,
            "createdAt": "2021-03-02T14:47:35.406+00:00",
            "orderId": "73b93f3d-8dee-4bd0-9a23-05d465be75f9"
        },
```

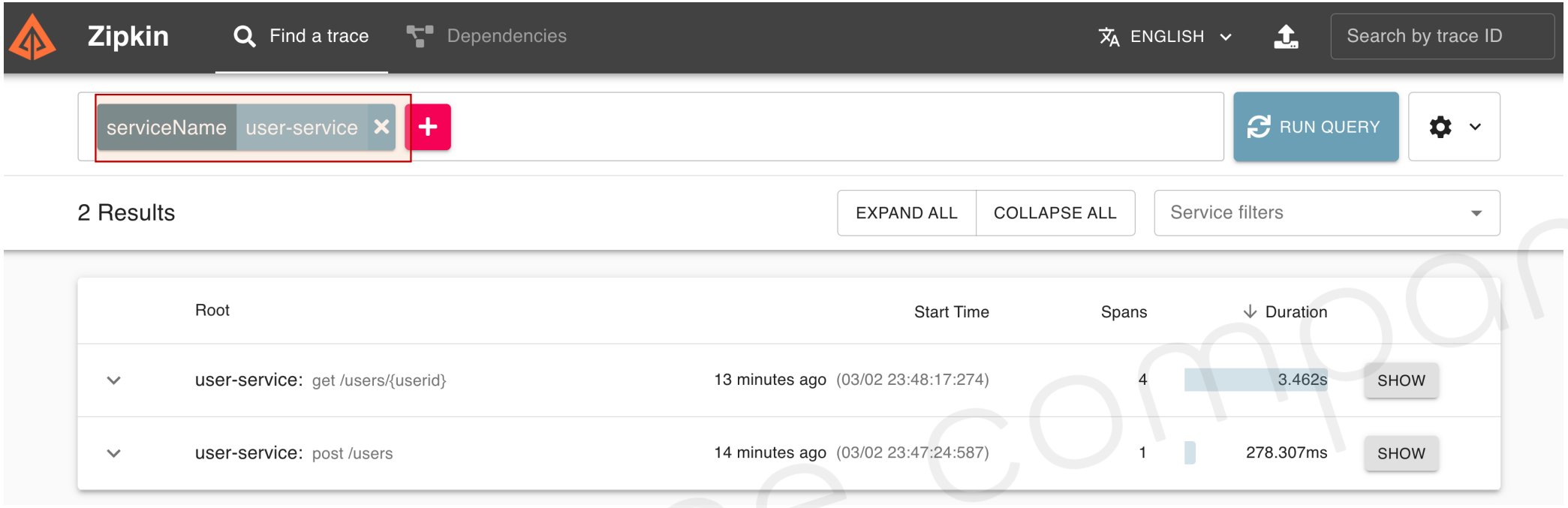
Trace ID and Span ID



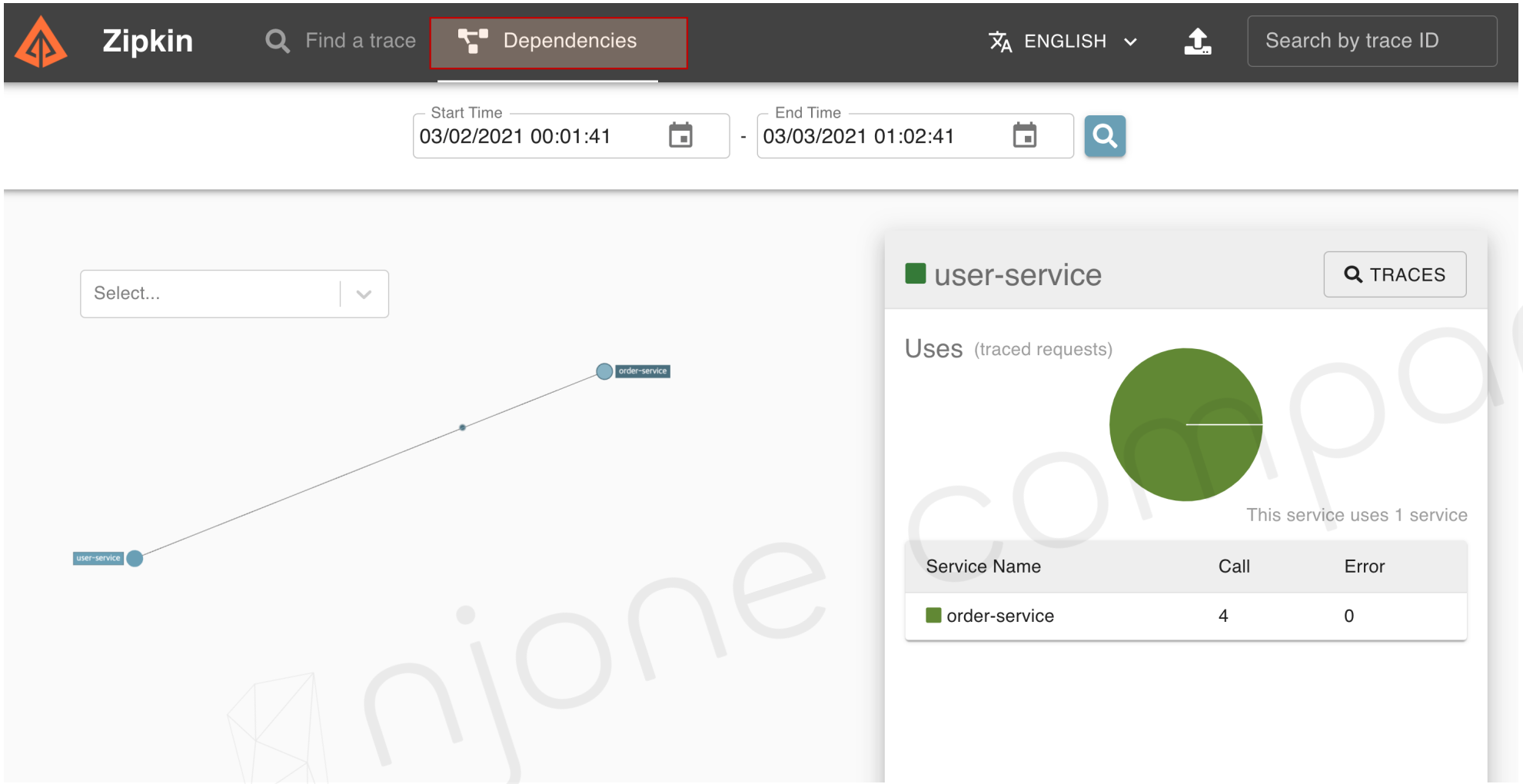
Zipkin server 활용



Zipkin server 활용



Zipkin 에서의 추적



Orders Microservice 장애 발생

```
@GetMapping(value="/{userId}/orders")
public ResponseEntity<List<ResponseOrder>> getOrder(@PathVariable("userId") String userId) throws Exception {
    log.info("Before retrieve orders data");
    Iterable<OrderEntity> orderList = ordersService.getOrdersByUserId(userId);

    List<ResponseOrder> result = new ArrayList<>();
    orderList.forEach(v -> {
        result.add(new ModelMapper().map(v, ResponseOrder.class));
    });

    try {
        Thread.sleep( millis: 3000);

        throw new Exception("장애 발생!");
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    log.info("After retrieved orders data");

    return ResponseEntity.status(HttpStatus.OK).body(result);
}
```

Orders Microservice 장애 발생

Orders Microservice 장애 발생